The ITSIMBW Environment for Simulation and Decision Support

Uwe Beyer, Thomas Zöller, Philipp Hügelmeyer and Timo Steffens
Fraunhofer Insitut for Intelligent Analysis- and Informationsystems
Schloss Birlinghoven
Sankt Augustin
Germany

uwe.beyer@iais.fraunhofer.de / thomas.zoeller@iais.fraunhofer.de /
philipp.huegelmeyer@iais.fraunhofer.de / timo.steffens@iais.fraunhofer.de

ABSTRACT

In this contribution, we give an overview on the military multi-agent simulation environment ITSimBw, which is developed at Fraunhofer IAIS/ART under contract for the department A5 of the IT office of the German Armed Forces (ITAmtBw), where the project is overseen by Captain Thomas Doll.

Due to the growing importance of network centric capabilities in military operations, one of the main focus points for the development of ITSimBw is the faithful modelling of IT and communication aspects. This goal is achieved essentially by two means:

1. *A message format for simulated communication acts between agents is provided which allows the detailed specification of communication channel, medium, and range.*
2. *A voxel-space representation is used to model the extension of all objects belonging to the simulated environment. This allows for the application of high performance ray tracing algorithms to precisely determine the impact of effects like radio-pockets (e.g. caused by mountains) as well as jamming by opposing forces.*

These two features in conjunction allow for a detailed and realistic modeling of communication chains for reporting and command both inside and across different echelons.

Clearly, the modelling of communication aspects can only reasonably be carried out in an environment which is rich enough to support the simulation of a broad variety of scenarios. ITSimBw addresses this point by strictly adhering to an agent-oriented paradigm which allows for the specification of autonomous, situation-based behaviour for all entities. This extended agent concept includes environment, weather, bridges, obstacles, and the like as active elements. This means that all effects and events are handled as actions of agents.

Another important issue for any simulation system is the precise and comprehensive description of the scenarios which are to be examined. To this end, ITSimBw encompasses its own LAMPS description language. Being based on high-level Petri-Nets, it can be represented graphically and by rule-sets. Moreover, due to its generality, it is equally well suited for the description of complete scenarios as for the specification of agent behaviour. An important area of ongoing research and development is the capability of LAMPS to record events. This feature enables the creation of a scenario data-base containing mission graphs from simulation runs as well as real-world manoeuvres or even actual military missions. This data-base can then form the core component of a decision support system for the military commander. Like a chess player comparing a current board position with memorized games to determine the next move, a graph-metric can be used to liken the LAMPS graph of an ongoing mission with those in the data-base. We thus envisage LAMPS to be a core factor for the application of data-mining-techniques in mission evaluation.

Beyer, U.; Zöller, T.; Hügelmeyer, P.; Steffens, T. (2006) The ITSIMBW Environment for Simulation and Decision Support. In *Transforming Training and Experimentation through Modelling and Simulation* (pp. 12-1 – 12-12). Meeting Proceedings RTO-MP-MSG-045, Paper 12. Neuilly-sur-Seine, France: RTO. Available from: http://www.rto.nato.int/abstracts.asp.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 SEP 2006	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The ITSIMBW Environment for Simulation and Decision Support	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Fraunhofer Insitut for Intelligent Analysis- and Informationsystems Schloss Birlinghoven Sankt Augustin Germany**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM002053., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**42**	

1.0 INTRODUCTION

In this contribution, we present the military multi-agent simulation environment ITSimBw and its description language LAMPS, which is used for the specification of agent behaviours as well as complete scenarios.

One of the main challenges of simulations is the optimal choice concerning the degree of model detail. Coarse models commonly yield very broad and abstract results with little value for the military practitioner. Overly detailed models on the other hand produce outcomes that are too dependant on minute situational details thus preventing their applicability to other scenarios even if they are highly similar. Therefore, scaling aspects play a major role in the design of simulation tools.

Another important issue for any simulation system is the precise and comprehensive description of the scenarios which are to be examined. To this end, ITSimBw [1] encompasses its own LAMPS description language. Being based on high-level Petri-Nets, it can be represented graphically and by rule-sets. Moreover, due to its generality, it is equally well suited for the description of complete scenarios as for the specification of agent behavior.

In our view, the unique combination of features outlined in the preceding paragraphs distinguishes ITSimBw from other commonly used military simulation tools such as MANA [2] or Pythagoras [3].

LAMPS can be used to document and analyze processes and to directly implement corresponding simulations. LAMPS is designed to have a broad application scope, so that a wide range of processes can be modelled, including business processes, warfare scenarios [1], critical infrastructures[4], or physical simulations. This wide applicability is due to LAMPS' generic approach which uses agents to represent all active entities. All entities in the system, including environment objects, IT-systems, or classic agents, can be modelled in the same way as agents. All effects and events are modelled as actions of agents.

In the next section we give a broad overview of related work. In Section 3 the key concepts of ITSimBw, which have already been touched upon in this introduction, are explained in more detail. Section 4 describes the concept of voxelspaces that are used to represent the spatial extent of agents. Section 5 then gives an exposition of the underlying architecture and the way communication is dealt with in the environment, followed by a presentation of the main concepts of the LAMPS description languageused for simulation and decision support, in section 6. Finally, section 7 concludes.

2.0 RELATED WORK

While languages for the specification of simulation models abound, LAMPS is to our knowledge the only one that supports both the description of scenarios and the executable specification of agent behaviour for compound agent groups down to individual agents. Existing simulation models are usually extensions of programming languages such as C/C++ (e.g. Maisie [5] or the SPaDES environment[6]) or Java (e.g. SILK[7], the SSJ package [8]). In contrast, LAMPS is based on high-level Petri nets [9]. Thus, LAMPS inherently supports parallel simulation, and is not an extension of sequential simulation languages like Maisie or SIMSCRIPT III [10]. Like other modern simulation languages [8], LAMPS can be displayed both graphically and as a rule-set.

LAMPS is based on hierarchical Petri Nets [9] and Colored Petri-Nets (cf. [11]). The former introduced the idea to structure so-called places (states) and actions hierarchically; the latter introduced the idea of typed tokens. LAMPS extends Petri-nets by the concept of agents.

ITSimBW extends classical agent-based simulation [12] by rigorously modelling all effects and events as actions of agents. Furthermore, ITSimBw handles scheduling by grouping agents on to different tagboards. A tagboard handles all messages for a group of agents.

3.0 THE AGENT PARADIGM

In contrast to other multi-agent systems, where a few complex agents interact in a feature-rich environment, ITSimBw is designed from the ground up with an agent-oriented paradigm in mind. Consequently, virtually every part of a simulation can be modelled as an autonomous agent. This includes humans, software, the landscape, weather phenomena like clouds or rain, and inanimate objects such as bridges and other buildings. Although many of these agents are usually passive, they can be activated at any time if required by the investigated scenario.

Since the principles of autonomous agents do not allow manipulating agents from the outside, all influences are communicated via messages to which the receiving agent is free to comply. Each agent consists of a set of attributes and an interface. The interface is used to communicate with the other agents. Effects like weapon fire, weather influences, and the like, are communicated via specialized messages which carry the effect type. The concrete impacts of effects, however, are stored in a central data structure, the effect table. Thus, the ITSimBw user has a single reference point for changing the impact of effects which can be influenced by non-programmers.

4.0 VOXELSPACE

Since scenarios in military operations are often heavily influenced by terrain, weather, the exact position of all objects, etc., an exact 3D representation of all physical entities is important for a realistic result of a military simulation. We use voxels to represent physical agents in our environment. A voxel is a portmanteau of the words volumetric and pixel. Each voxel represents a regular volume of space. Contrary to the commonly used approach we organize voxels inside an octree. Octrees are usually used to subdivide a scene into elements (in this example boxes), so that each element ideally contains one object of the scene. So in a regular octree, a box is divided into 8 equally sized subboxes, which than again can be divided into 8 subsubboxes and so on.

Since we have not a static scene, where all objects stay in their boxes, we use a voxelspace for each individual object. This is also used in modern computer games like Delta Force or Command & Conquer. We use octrees, because insertion and deletion of objects are cheap (O(1), if one knows the size of the new object), there are efficient algorithms to intersect cubes and lines (the use will be shown in the next section) , and voxelspaces can be efficiently used for occlusion culling. In addition each voxel may contain additional attributes like density, weight, and so on. There are also different densities for different effects, which may be influenced by a voxel, e.g. for radio transmissions, vision, and physical density (if one can move through an object).

5.0 THE FTA ARCHITECTURE

5.1 Core Concepts

The ITSimBw simulation environment is based on the Flip-Tick-Architecture (FTA) [13], which has its roots in the JANUS project developed at the *Gesellschaft für Mathematik und Datenverarbeitung (GMD)* [14]. At its core, FTA is a design paradigm for scalable distributed systems that exhibit unknown dynamic characteristics as well as disturbances and inaccuracies which are difficult, if not impossible, to model in a closed-form mathematical approach. As every multi-agent system, FTA is based on the concept of a society of agents [15]. It comprises four classes of entities: actors (i.e. agents), assemblies, tags, and tag-boards.

The aforementioned society of agents *A* is formed by a set of individual agents a_j, which are called actors in FTA terminology:

$$A = \{a_1, ..., a_j, ...a_k\}. \quad (1)$$

Each actor a_j is composed of a set of typed attributes U_j, whose value assignment determines the agent's state, together with an action function, which entails all operations that can be performed by the actor.

$$a_j = (U_j = (u_{1,j}, ..., u_{m,j}), f_j). \quad (2)$$

Structural information concerning agents, i.e. names and types of attributes, is described via agent types. Formally, we have $type(a_j) = t$, if and only if agent a_j is of type t. The system supports agent templates that can be used to store prototypical value assignments. Thus, an individual agent can be created either by instantiation of its type, or by copying from a pre-defined template.

The principle of autonomy of agents forbids the direct manipulation of internal data structures and behaviors of other agents. Consequently, all interactions between agents are handled via messages. Formally, a message N_v is comprised of a number of attributes:

$$N_v = (u_{1,v,...,} u_{l,v}). \quad (3)$$

As agents, messages are typed entities. The set of all message types is given by

$$N = \{N_1, ..., N_h\}. \quad (4)$$

Upon receiving a request, the agent is able to analyze its content and to decide whether it wants to comply. The basic unit of execution is called a *cycle*. During one cycle, the agent reads its messages and triggers the appropriate actions, which might consist of writing messages to other actors.

An assembly is a set of agents sharing a common pace, i.e. all elements of an assembly have the same time resolution *dt*. This in turn implies that their cycles are synchronized and that the assembly switches from cycle to cycle as regularly as the tick of a clock. Hence the term *Tick* in FTA. It is important to note, that different agents do not necessarily share the same time resolution. Instead, the architecture supports individual running speeds for every actor. Moreover, time steps can vary from cycle to cycle. Thus, adaptive control of time increments can be realized (see below). This is particularly valuable for increasing the time resolution in the computation of dynamics equations for fast moving objects.

The messages used for inter-agent communication are called *tags* in FTA. We distinguish between three different categories of tags:

- Message tags are the carriers of communication between agents. Orders are embedded in message tags using XML syntax.
- Position tags contain the current position of agents. They are sent when actors change their position.
- Effect tags are used to mediate effects like weather influences, weapon impact, and the like.

Instead of setting up a direct communication with other actors, agents register with one ore more *tag-boards*, to which they send their messages. Thus, tag-boards serve as the functional units for handling messages in the FTA system. In formal terms, a tag-board forms a medium M_n for message exchange, while a FTA system is capable of supporting multiple media:

$$M = \{M_1, ..., M_m\}. \quad (5)$$

A tag-board consists of two sides. One is write-only and contains all tags sent to the board in time step *t*, whereas the other side is read-only and encompasses all tags written in time-step *t*-1. Analogously to agents, each tag-board has its own time resolution and thus its own cycle time. During a board cycle, the write-only side is flipped over. Thereby, the read-only part mirrors the tag content of the write part from the previous time-step. The write-only side is deleted after flipping. In this way, the lifetime of tags is effectively controlled by the time-scale of the pertaining board. These interrelations are depicted graphically in figure 2.

With this approach, fully synchronized (all agents and tag-boards share the same time resolution) as well as completely asynchronous systems (every agent and every board has its own time-scale) can be modeled in terms of the FTA.

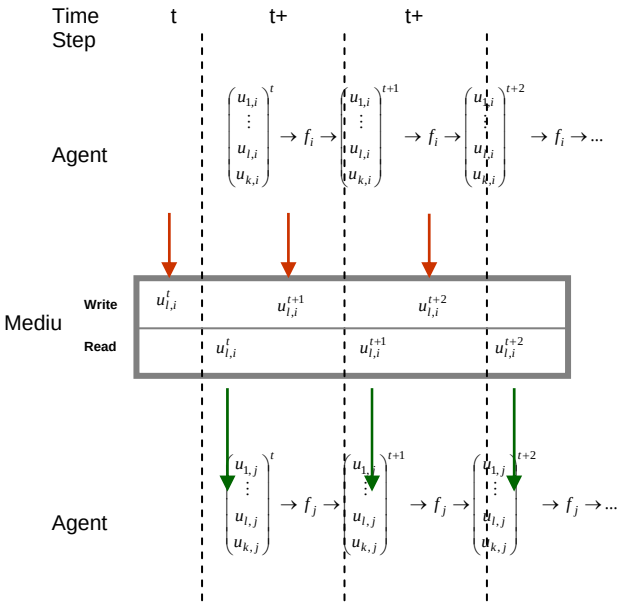


Figure 1: Inter-agent communication via tag-board.

The mathematical model of this agent architecture is a system of flexibly coupled inhomogeneous difference equations

$$D = (D_1, ..., D_k), \qquad (6)$$

where each agent computes an equation from D. In this setting, the attribute list of an actor corresponds to the variable vector of the pertaining difference equation. Thus, equation D_i of agent a_i computes

$$U_i^{t+dt_i} = f_i(U_i^t), \qquad (7)$$

where dt_i denotes the time step size of agent a_i. During the iteration of D, the society of agents runs through a set of states $Z = \{Z^t\}$, where each of these system states is the union of attribute sets from all participating agents:

$$Z^t = (U_1^t, ..., U_i^t, ..., U_k^t). \qquad (8)$$

Since the D_i are computed using potentially different dt_i, the involved agent attribute sets U^t_i might be undefined for certain t. In this case, either the U_i from the last time step of D_i is used, or U_i is explicitly recomputed. We thus have

$$Z = Z^0, ..., Z^{t_i}, Z^{t_j}, ..., Z^e$$

where $$t_j - t_i = \underset{h=1}{\overset{k}{Min}} dt_k. \quad (9)$$

A graphical visualization of FTA (omitting the discrimination of different tag-board sides) is shown in figure 3:

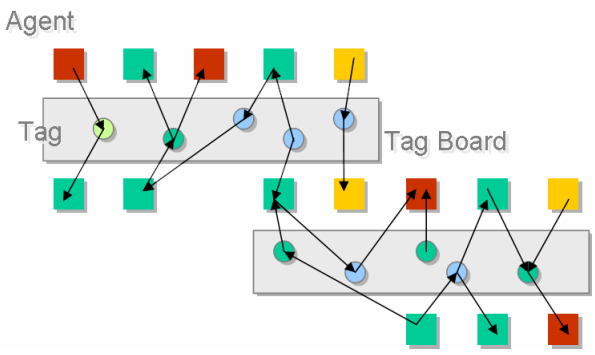


Figure 2: The Flip-Tick-Architecture. Arrows denote read() and write() operations. The colors specify types of agents and tags. It is depicted that agents can subscribe to more than one tag-board.

5.2 Communication

Although ITSimBw does not force its users to model IT and communication equipment as individual agents, this procedure is highly encouraged by the pervasive agent-based modeling paradigm.

The simulation system supports communication modeling essentially by two means:

- A message format for simulated communication acts between agents is provided which allows the detailed specification of communication channel, medium, and range.
- The voxel-space mentioned in the previous subsection is used to determine environmental effects on communication, e.g. mountains causing radio pockets, as well as jamming by opposing forces.

These two features in conjunction allow for a detailed and realistic modeling of communication chains both inside and across different echelons.

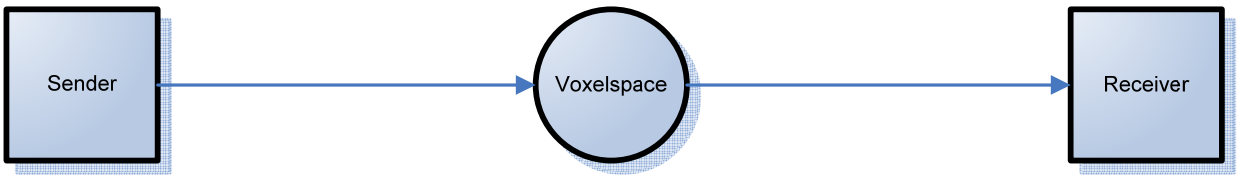


Figure 3: The model of a Message send from Sender to Receiver through a Voxelspace

6.0 LAMPS

6.1 Basic Constructs

LAMPS uses the following concepts from hierarchical and coloured Petri nets:

- Places hold states, i.e. in LAMPSSys they contain subsets of the agent attributes. The content of a place is called token and can be of an arbitrarily complex structured type. Places can contain several tokens of different or identical types.
- Actions describe the effects that agents apply to themselves or to other agents.
- Relations denote the links between places, actions, and agents. Relations correspond to the arcs in the Petri net model.

LAMPS introduces the additional concept of agents in the following way:

- Agents correspond to the conditions of Petri nets. An agent in LAMPS observes the set of places that have relations to the agent's actions. Based on these places the agent decides which actions are executed and which parameters should be used.

Agents extend guard functions (Esser 1997) and enabling functions (Schoef 1995). These serve as preconditions that are checked before an action is executed. Figure 4 depicts a simple example using the four basic concepts.

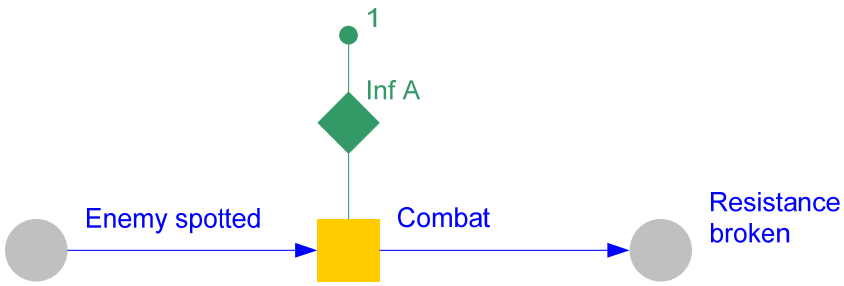


Figure 4: A LAMPS fragment using the four basic concepts. The agent *Inf A* observes the place *Enemy spotted*. If the place contains a token, the agent executes action *Combat*, which sends a token into the place *Resistance broken*.

LAMPS is inherently parallel due to its core of high-level Petri nets. Agents live in parallel, and each agent can execute several parallel actions. All actions whose conditions are true in a given cycle are executed. This is also the main difference to flow-charts, because there are several tokens per place, and several places can be filled simultaneously.

All basic constructs can be recursively encapsulated. For example, places can be combined to so-called superplaces. An action can be recursively defined as a LAMPS process (with places and other actions), as long as the interface (i. e. the incoming and outgoing places) of the action and the process are identical. This way, a process can be modelled and viewed on different detail levels.

Since LAMPS introduced agents, also agents can be recursively encapsulated. A group of agents can be aggregated into an agent (say, a group of soldiers into an infantry unit). The level of detail can be modulated even at runtime (see section below).

6.2 Features

While LAMPS is not restricted to a time model, LAMPSSys is discrete-time and thus discrete-event.

Time-modelling is not trivial in high-level Petri-nets (cf. [11]). Thus, in the LAMPSSys framework the execution is cycle-based with a global clock. In each cycle all agents can execute one or more actions. Long actions are modelled as a series of consecutive actions of duration *dt*. Synchronisation is achieved via places and conditions.

In the approach, time scaling is possible. In each cycle, each agent proposes a duration for its action. The simulation engine selects the minimum of the proposed durations and sets *dt* accordingly. To illustrate this, assume that an agent is about to execute an action that needs a coarse granularity. This could be for example an action that moves a soldier from one position to a distant position, proposing $dt = 5min$. Assume that another agent is about to execute a more detailled action in the time-cycle, for example, shooting at an enemy, proposing $dt = 0.01sec$. In this case, the simulation will set the general *dt* to $0.01sec$, so that every action in this time cycle is executed in this granularity.

6.3 Graph-Metric

In a simulation the time is run in disc cycles. Time is therefore a series' of timestamps:

$$t_1 \rightarrow t_2 \rightarrow ... \rightarrow t_m$$

A state S in a point in time t is described by the states of all the individual agents s1 to sn.

$$S_t = \{s_1, s_2, ..., s_n\}$$

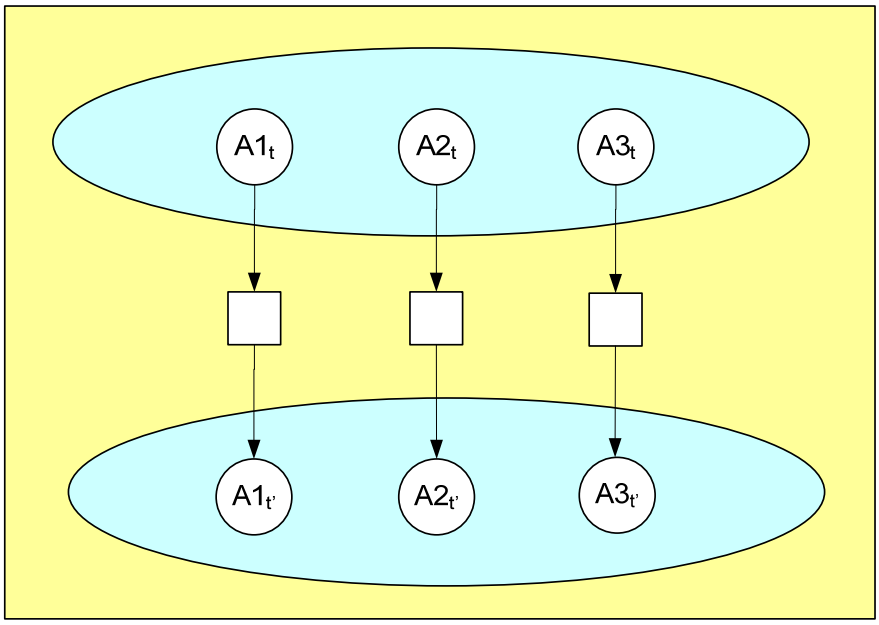


Figure 5: The state of each agent is transformed from Time t -> t' by actions performed on these states.

A simulation run is fully described by the set of all states S in all time steps t. Repeatedly running a simulation with varying parameters leads to a set of results:

$$R = \{r_1, r_2, ..., r_h\}$$

Our target is to find out typical situations over several runs. Since all states of all agents in all time steps result in an overwhelming amount of information, certain attributes that are considered relevant are chosen beforehand. For military simulations these are usually:

- The position (in the 3D space)
- The loss of own troops
- The time to accomplish the mission goal
- The amount of effects to act upon the opponent

With these four constraints and the time at which a certain state is reached, we then build equivalence classes over different runs. This can be accomplished by k-means clustering.

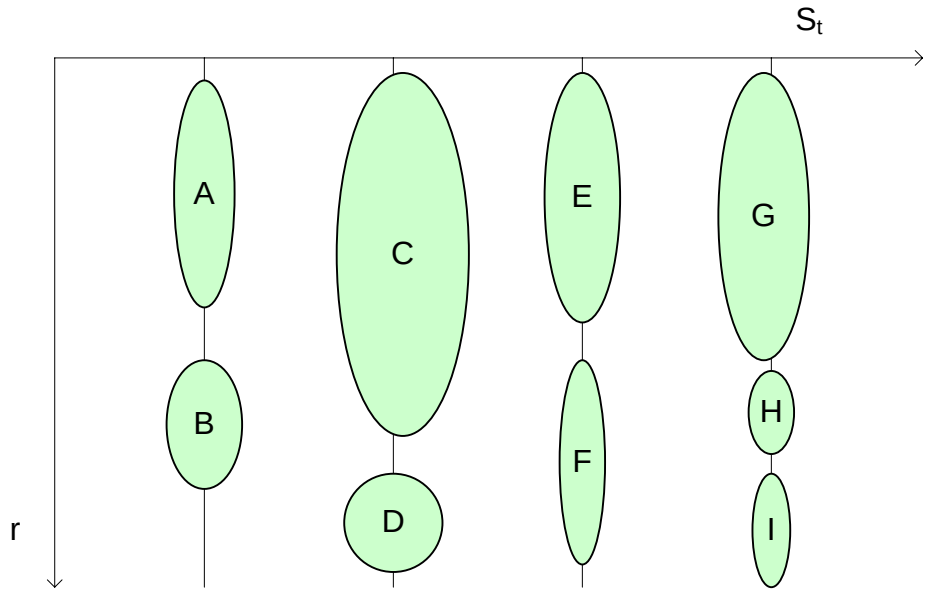


Figure 6: Equivalence classes over time and runs.

With these generated clusters we can build a Markov chain that has the probability that a certain state can be reached from the actual state out of the cluster.

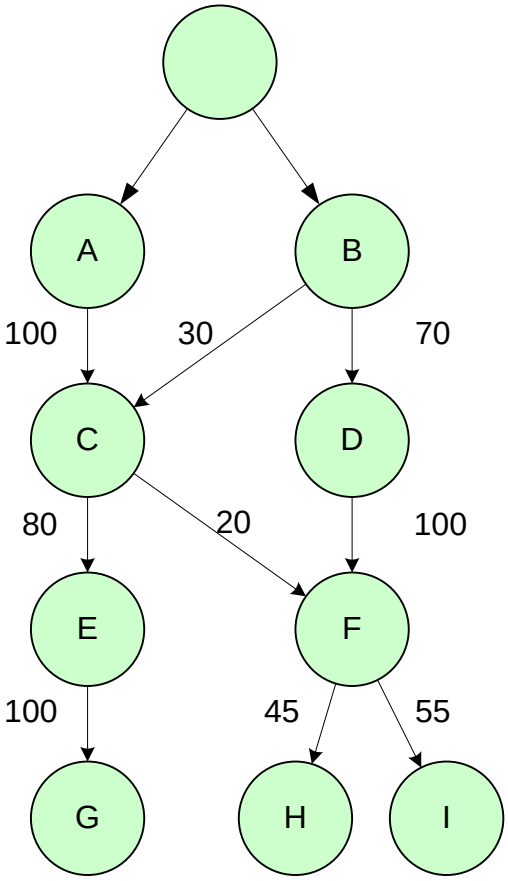


Figure 7: Markov chain describing the probabilities of transformation

7.0 CONCLUSION

In this paper we have given a broad overview over our architecture. We have shown the basics of LampsSys and how we gain information about our simulated task by multistarting our simulation and then cluster over the resulting runs.

8.0 REFERENCES

[1] Hügelmeyer P.; Steffens T.; and Zoeller T., (2006). *Specifying and simulating modern warfare scenarios with ITSimBw*. In L.F. Perrone; F.P. Wieland; J. Liu; B.G.L.D.M. Nicol; and R.M. Fujimoto (Eds.), Proceedings of the 2006 Winter Simulation Conference.

[2] Stephen, R. T., Anderson, M. A. and Lauren, M. K. 2002. MANA Map Aware Non-uniform Automata Version 2.0 User's Manual.

[3] Bitinas, E. 2002. *Pythagoras: The newest member of the Project Albert family. Computing Advances in Military OR* – WG 31. 70th Military Operations Research Society Symposium, Ft. Levenworth, June 18–20.

[4] Flentge F. and Steffens T., 2006. *IRRIIS - A new European Project to Increase CII Dependability*. European CIIP Newsletter, 4.

[5] Bagrodia R.L. and Liao W.T., 1994. *Maisie: A Language for the Design of Efficient Discrete-Event Simulations*. IEEE Transactions on Software Engineering, 20, no. 4, 225-238. URL citeseer.ist.psu.edu/bagrodia94maisie.html.

[6] Teo Y.; Tay S.; and Kong K., 1998. *Structured Parallel Simulation Modeling and Programming.* In Proceedings of the 31st Annual Simulation Symposium. IEEE Computer Society Press, 135-142. URL citeseer.ist.psu.edu/teo98structured.html.

[7] Healy K.J. and Kilgore R.A., 1997. *Silk: A Java-based Process Simulation Language*. In Winter Simulation Conference. 475-482. URL

[8] L'Ecuyer P. and Buist E., 2005. *Simulation in Java with SSJ*. In Proceedings of the 2005 Winter Simulation Conference. 611-620.

[9] Jensen K., 1992. *Coloured Petri Nets: Basic concepts, analysis methods and practical use*, Monographs in Theoretical Computer Science, vol. 1: Basic Concepts, Springer, Berlin.

[10] RICE S.V.; Marjanski A.; Markowitz H.M.; and Bailey S.M., 2005. *The Simscript III Programming Language for Modular Object-Oriented Simulation*. In Proc. Of the 2005 Winter Simulation Conference. 621-630

[11] Vojnar T., 1997. Various Kinds of Petri Nets in Simulation and Modelling. In J. Stefan (Ed.), Proc. of 31st Spring International Conference on Modelling and Simulation of Systems MOSIS'97. vol. 1, 227{232. URL citeseer.ist.psu.edu/vojnar97various.html.

[12] Moss S. and Davidsson P., 2001. *Multi-Agent-Based Simulation*. Springer, Berlin.

[13] Richter, G. 1999. Flip-tick architecture: A cycle-oriented architecture for distributed problem solving. In *GMD Resport # 19*. GMD.

[14] Beyer, U. and Smieja, F. 1994. Janus: A society of agents. In *GMD Report # 840*. GMD.

[15] Weiss, G., ed. 1999. *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge MA.

The ITSIMBW Environment for Simulation and Decision Support

Fraunhofer Institut
Intelligente Analyse- und
Informationssysteme

Philipp Hügelmeyer

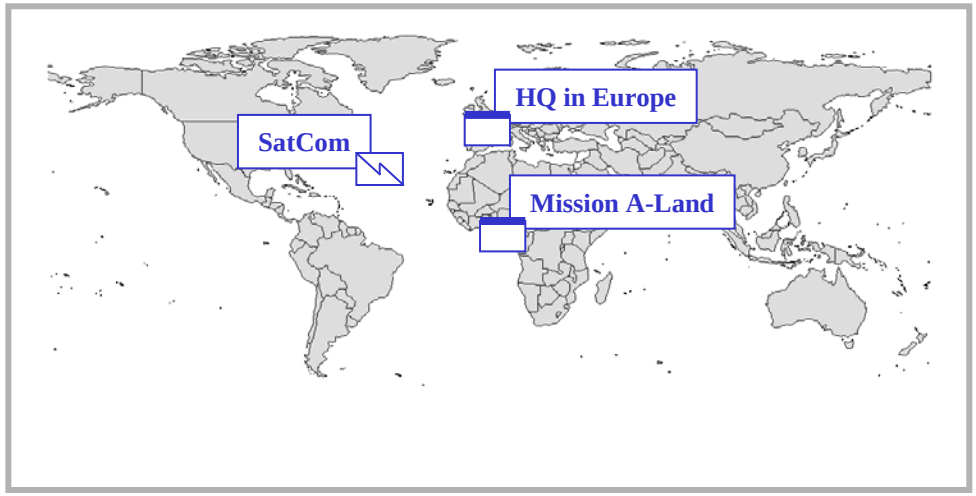
Dr. Timo Steffens

Outline

- Features of ITSimBW
  - Scalability
  - Focus on Cummunication
  - Freedom of modelling
  - Viewing alternatives graphically (decisions and/or events)
- Basic principles of technical realization
- Language for Agent-based Modelling of Processes and Scenarios (LAMPS)
- Graph Metrics
- Roadmap for the next steps (Gis, HLA …)

Scalability Multi-Scale Approach

Strategic level



- Global view
- Mission troups act as one single agent
- Time scale 1 minute

Tactical level

- Local effects are modeled
- More detailed agents
- Time scale 1 second

Scalability Multi-Scale Approach

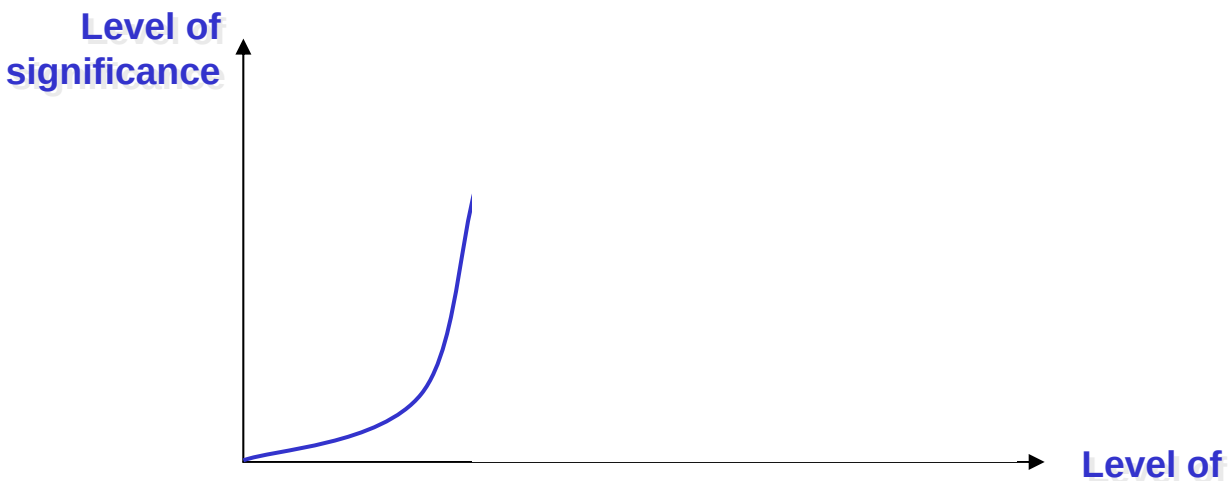
- Scalable complexity:
  - functionality (how detailed are functions/effects modeled/simulated ?)
  - time (in which time steps will the simulation proceed ?)
  - space (which spacial scale will be chosen for modeling?

- changable during run time
- at different places different changes possible
- system can be built in several stages (also depending on modeling efforts)

- Granularity in modeling should be further refined only if it makes sense

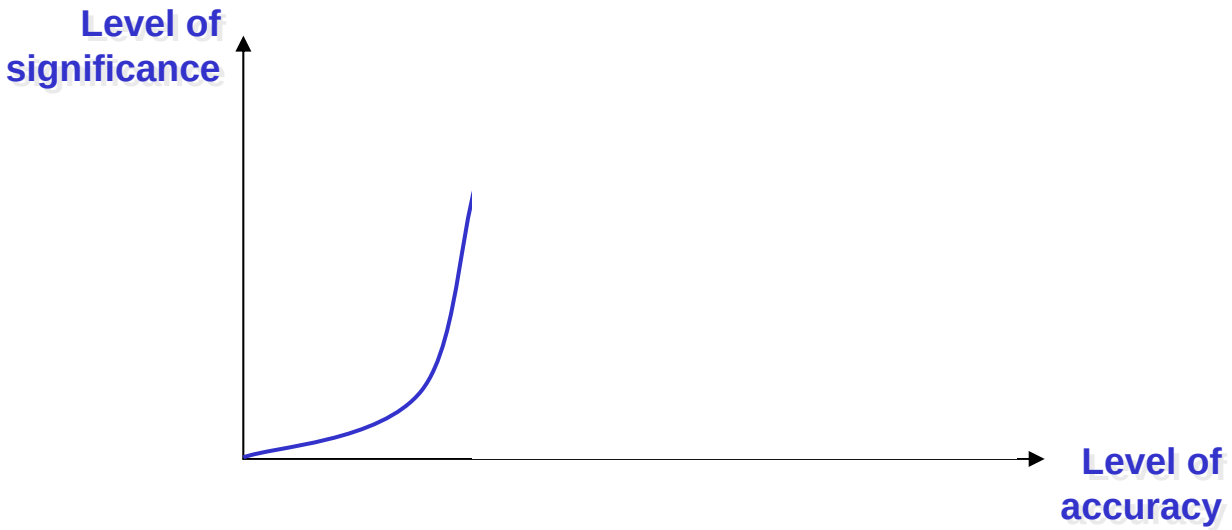
Scalability which level of accuracy is needed?

Scalability which level of accuracy is needed?



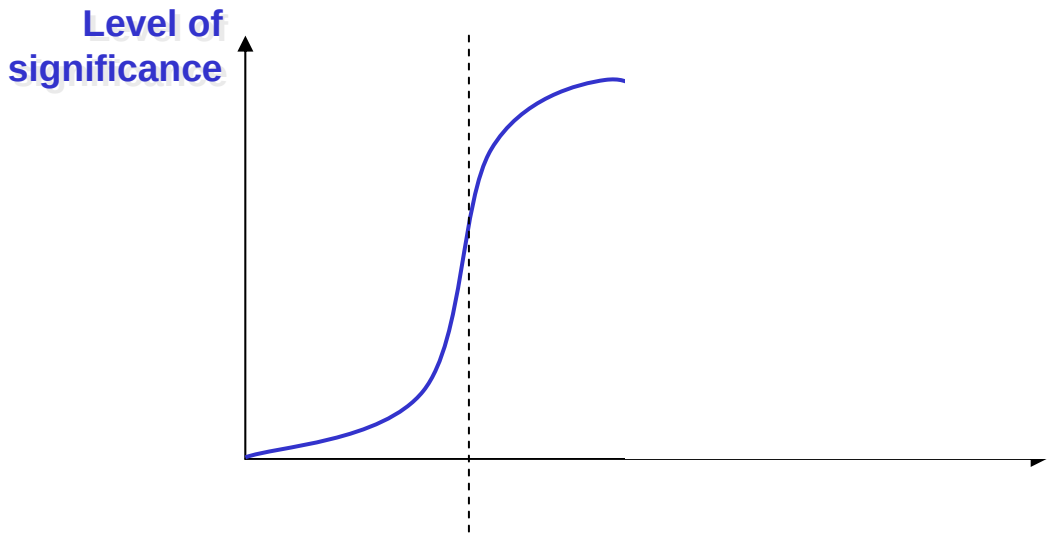
Assumptions/ models	Complexity	low
	Plausibility	very high

Scalability which level of accuracy is needed?



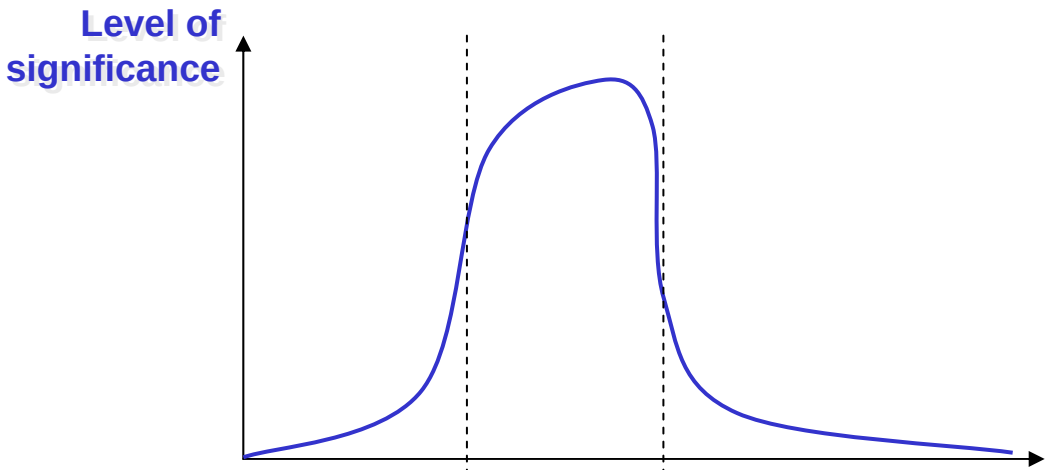
Assumptions/ models	Complexity	low
	Plausibility	very high
Results	Correctness	very high
	Usefulness	low

Scalability which level of accuracy is needed?



Assumptions/ models	Complexity	low	high
	Plausibility	very high	high
Results	Correctness	very high	high
	Usefulness	low	high

Scalability which level of accuracy is needed?



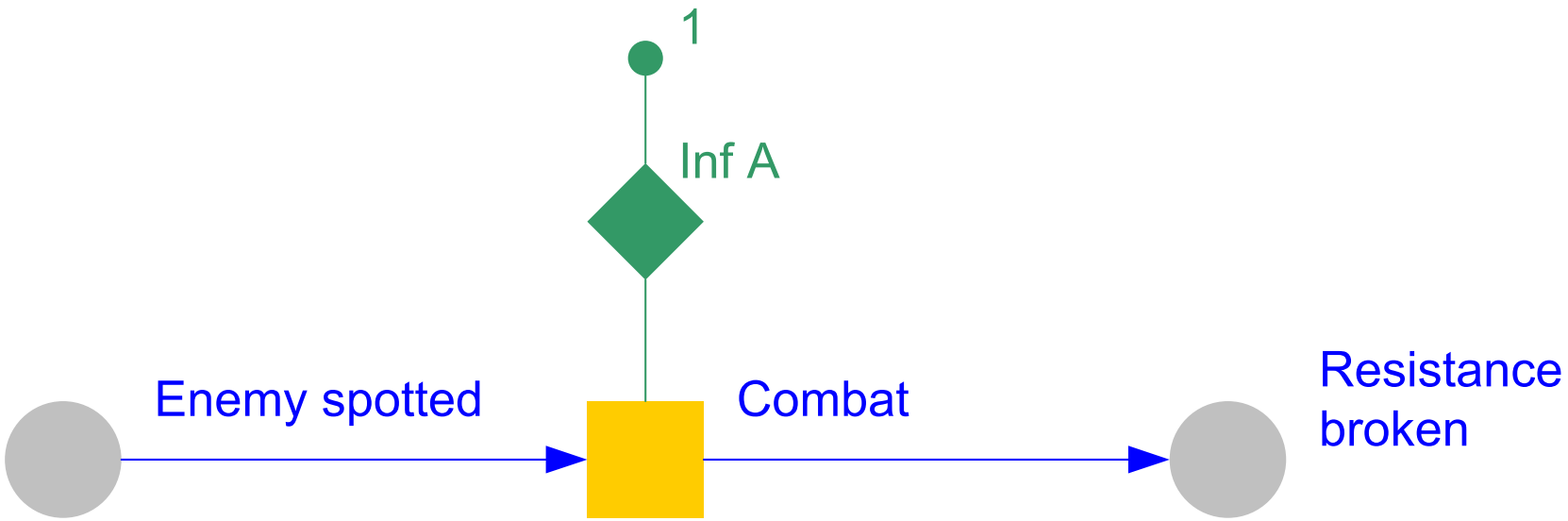
Assumptions/ models	Complexity	low	high	extremely high
	Plausibility	very high	high	low
Results	Correctness	very high	high	low
	Usefulness	low	high	low

- A simulation model must be adjustable to the appropriate scope
- This scope is changing from task to task

LAMPS

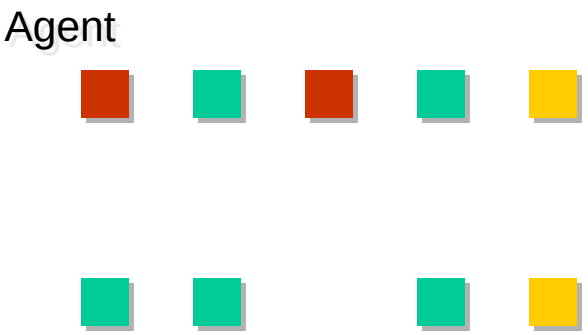
- Language for Agent based Modelling of Processes and Scenarios
- Based on high-level Petri-Nets
  - Actions describe the effects that agents apply to themselves or to other agents
  - Relations denote the links between places, actions, and agents.
  - Agents correspond to the conditions of Petri nets
- Graphical and textual representation
- Not restricted to a timing model
- Time scaling is possible

LAMPS



Agent Architecture FTA

What are the active system components ?



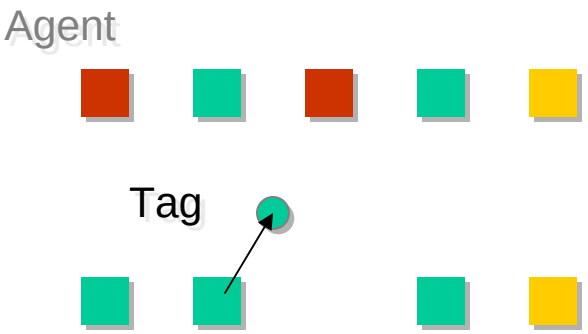
Autonomous active elements

- Persons
- Machines, vehicles
- Objects, Bridges, …
- …

Agents carry out the system processes

Agent Architecture FTA

How do active elements interact ?



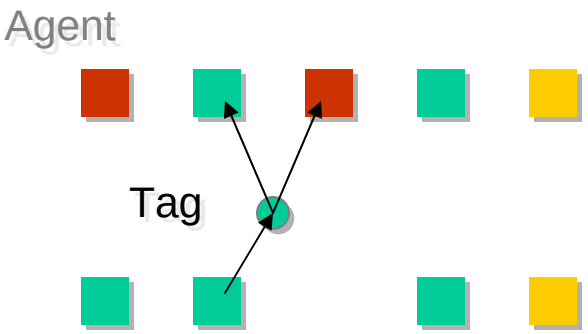
Causing effects

- spoken messages
- physical interaction
- effects of weapons
- …

Agents cause effects by producing tags

Agent Architecture FTA

How do active elements interact ?



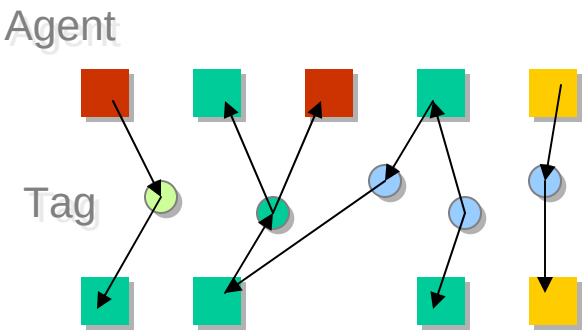
Taking effects

- reading a message
- realizing physical effects
- taking orders
- …

Agents are self-responsible for their own behavior, there is no direct action from outside

Agent Architecture FTA

How do active elements understand each other ?



Tags have a type

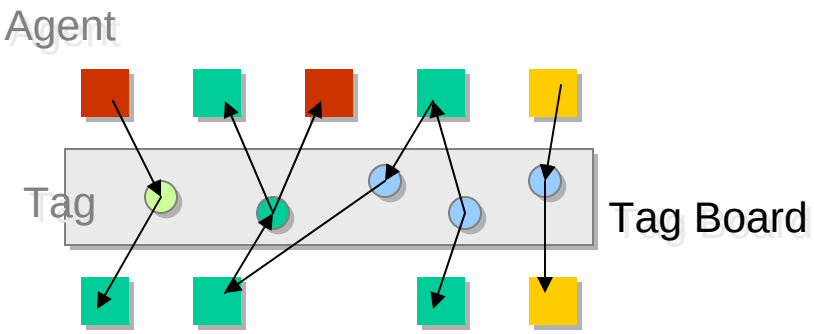
- order form
- general message
- physical power (effects)

- …

Tag types describe the form, tag instances contain the concrete data

Agent Architecture FTA

What is the radius of an interactionen ?



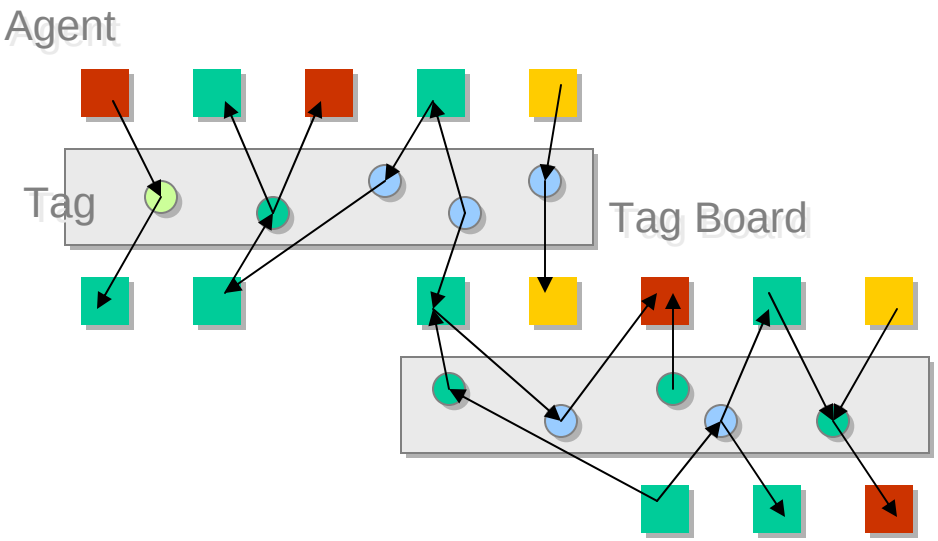
Regions of local interaction

- rooms, streets
- radius of physical effects
- communication links
- ...

Tag boards serve as models for regions of interaction

Agent Architecture FTA

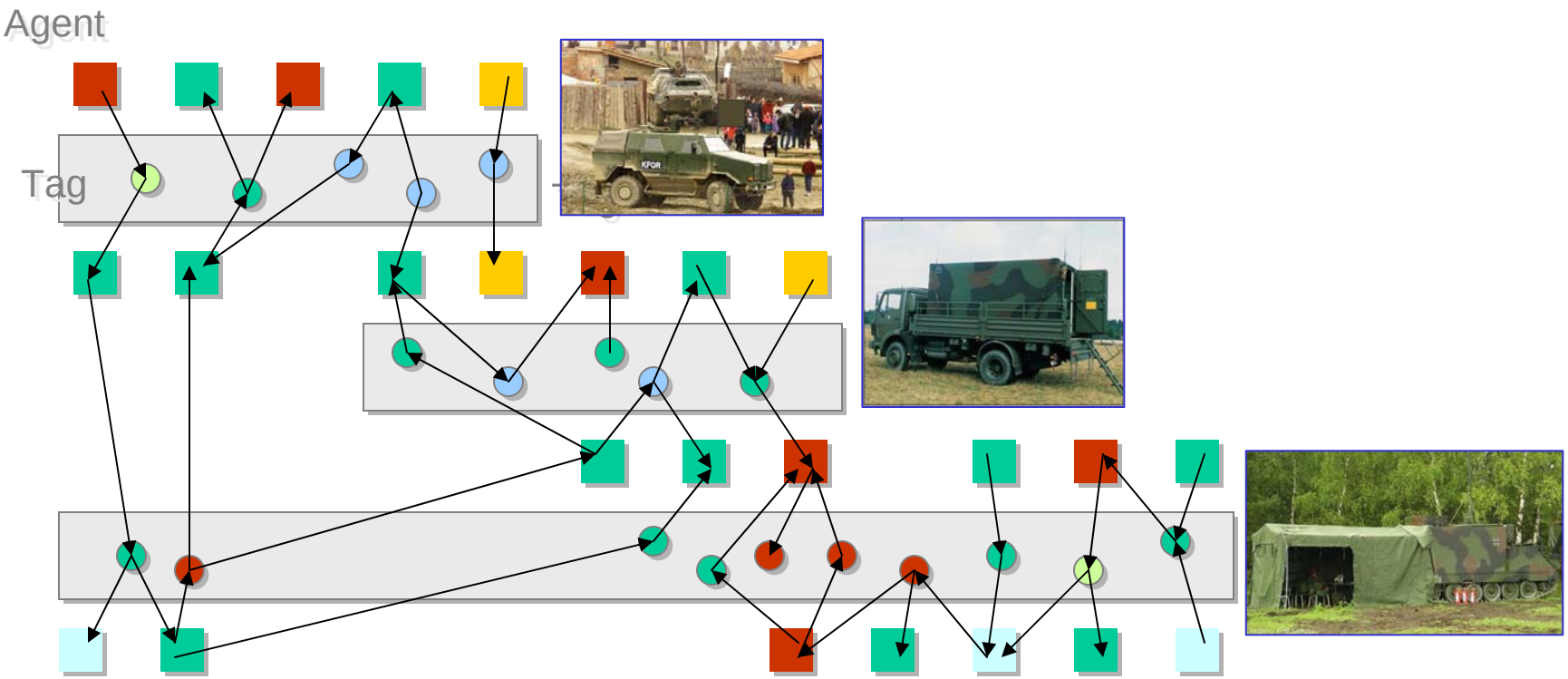
What is the system‘s topology ?



Agents can link together several regions of local interaction

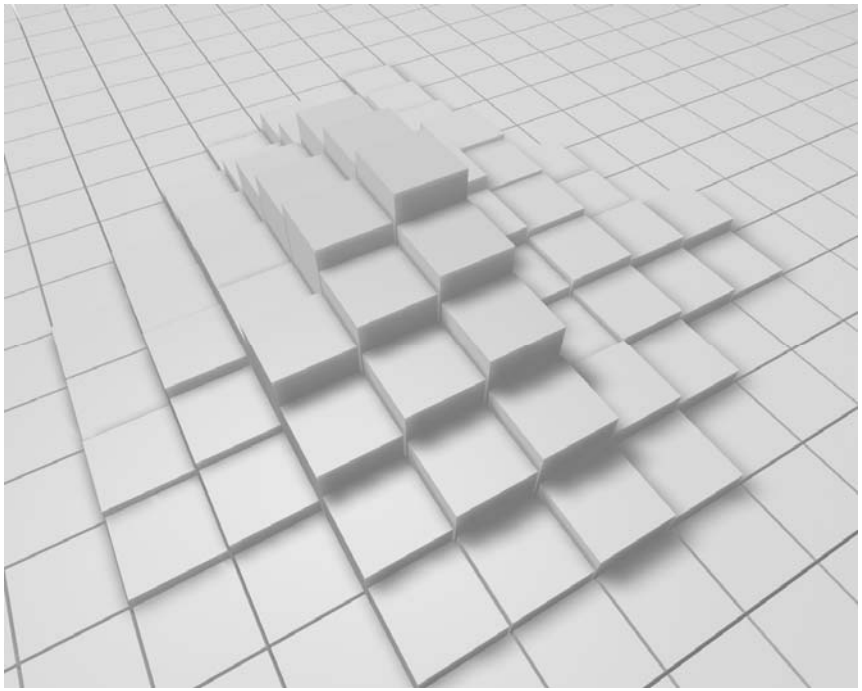
Agent Architecture FTA

What is the system‘s topology ?



Real topologies can be modeled 1:1

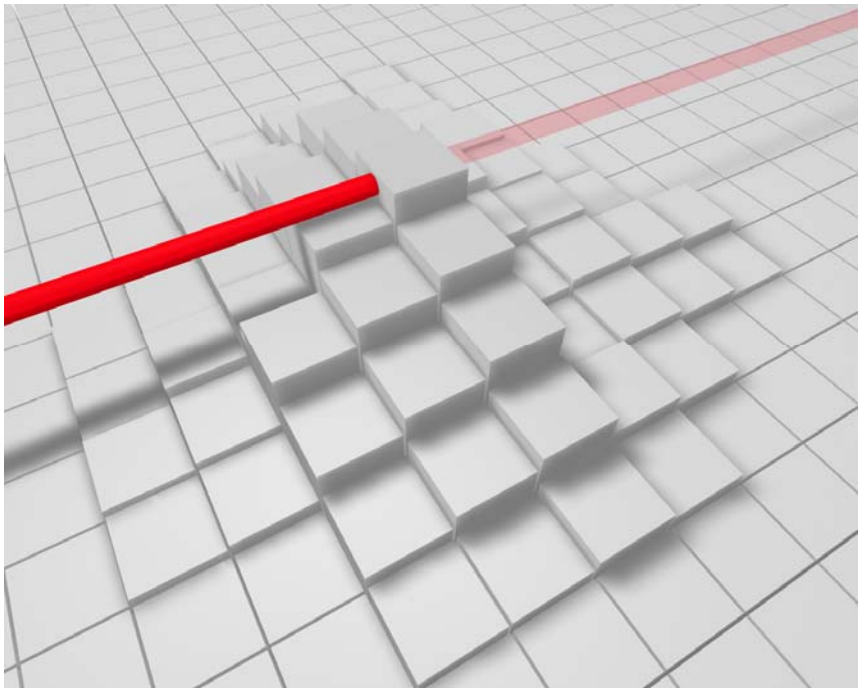
Modelling Environment



landscape model

- 3D voxel model for environment

Modelling Environment + Communication

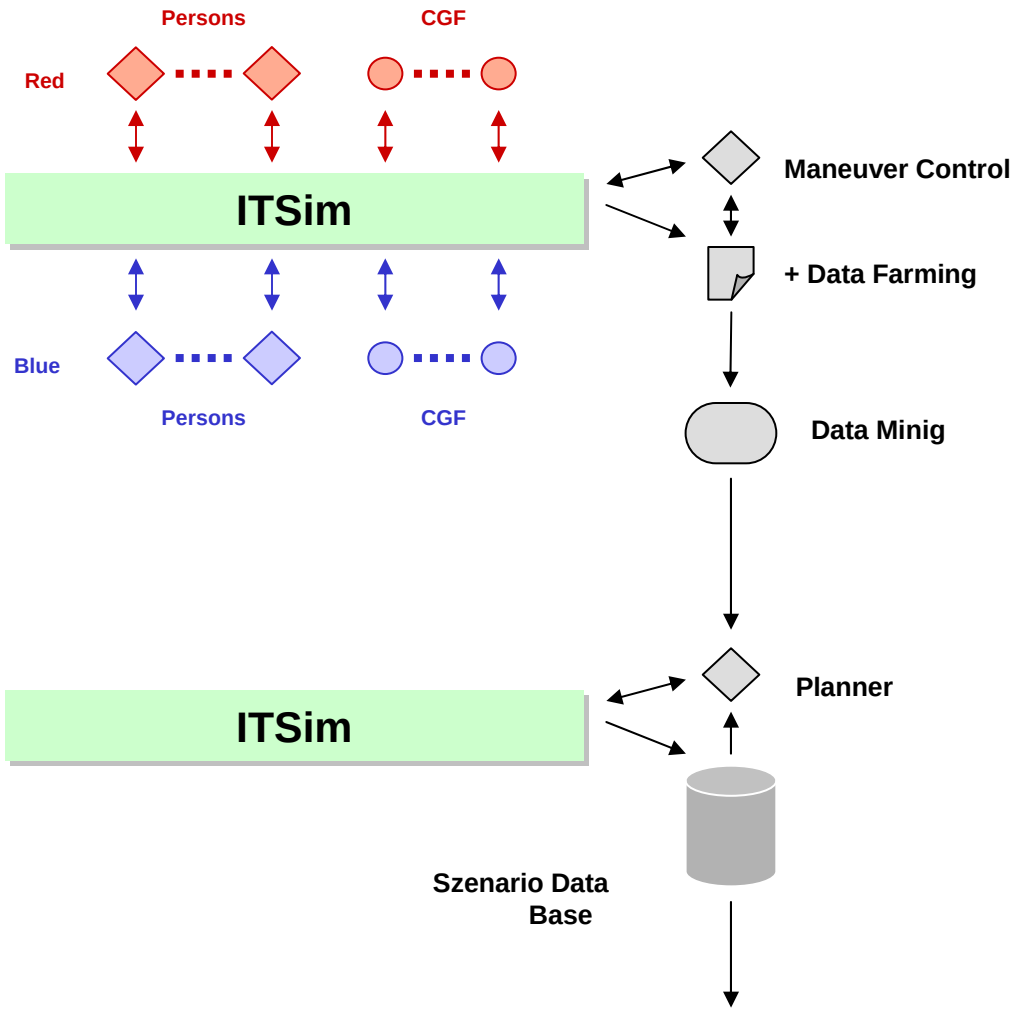


landscape model with radio beam at a hill

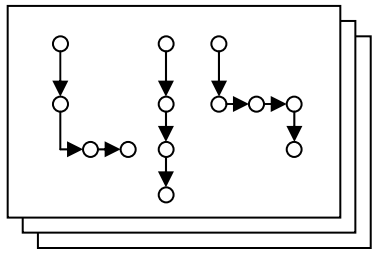
- 3D voxel model for environment
- obstruction of radio communication
- electronic countermeasures
- weather, clouds, mist, smoke
- visibility
- shapes of units
- ...

- computationally fast
- scalable in resolution
- appropriate for multi-scale features

Modes of use Nondeterministic Wargaming

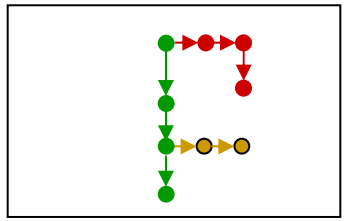


1. Maneuver mode



Variety of possible traces for different outcomes

2. Get empirical knowledge about the szenario

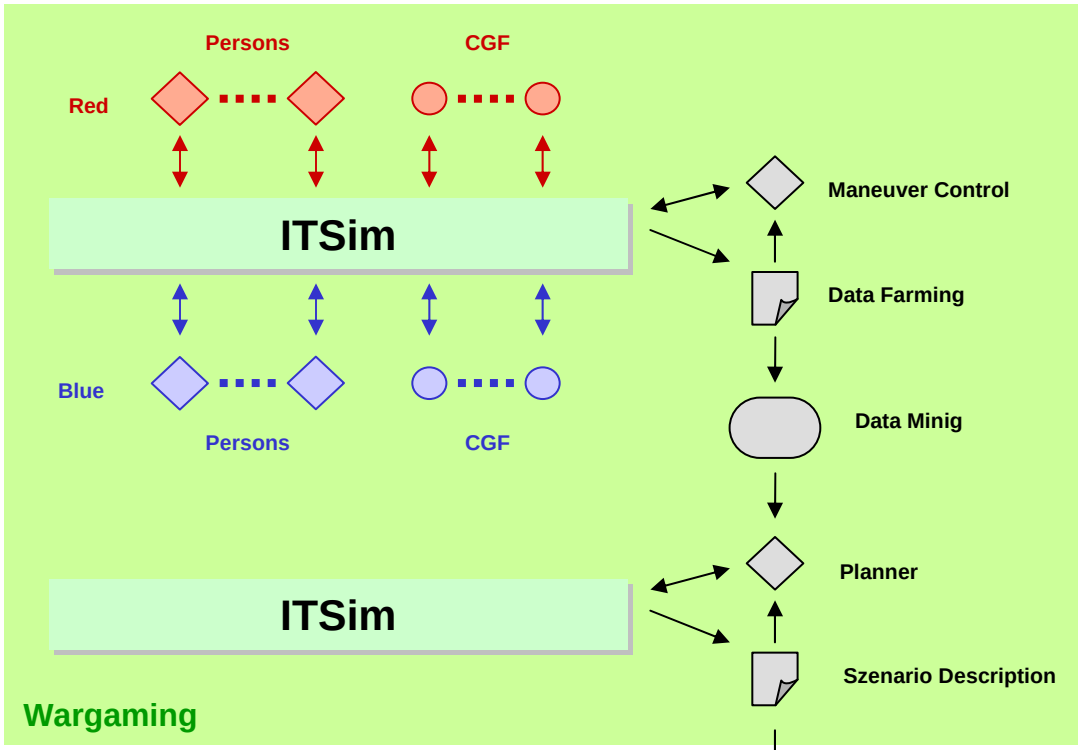


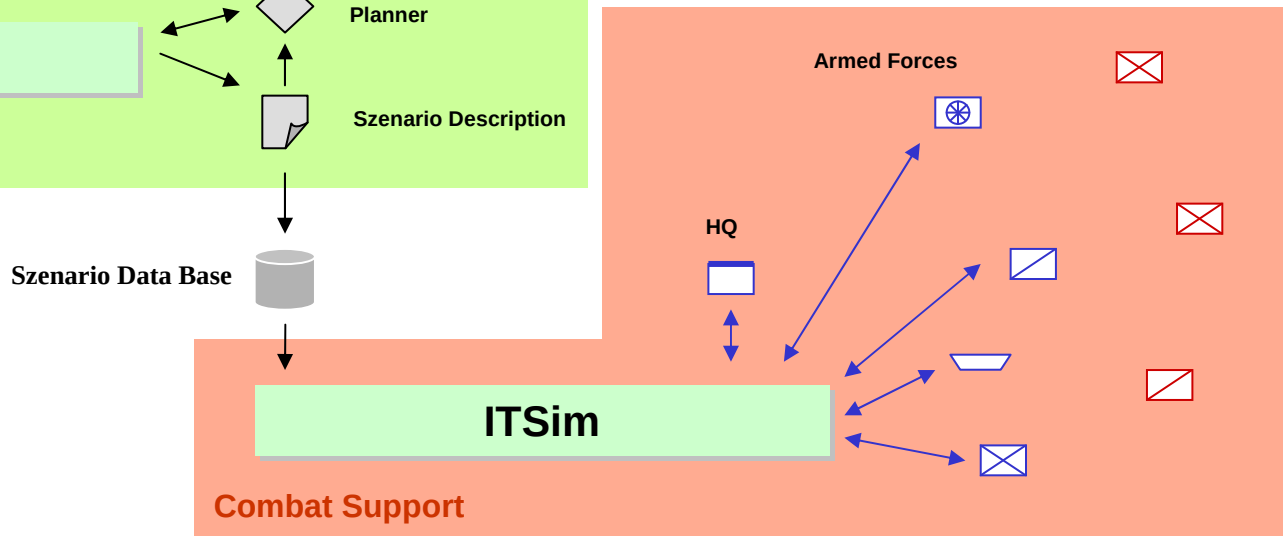
Evaluated non-deterministic szenario

3. Planner / Strategy-Builder
4. Szenario Data Base

„To know what might happen"

Modes of use Combat Support



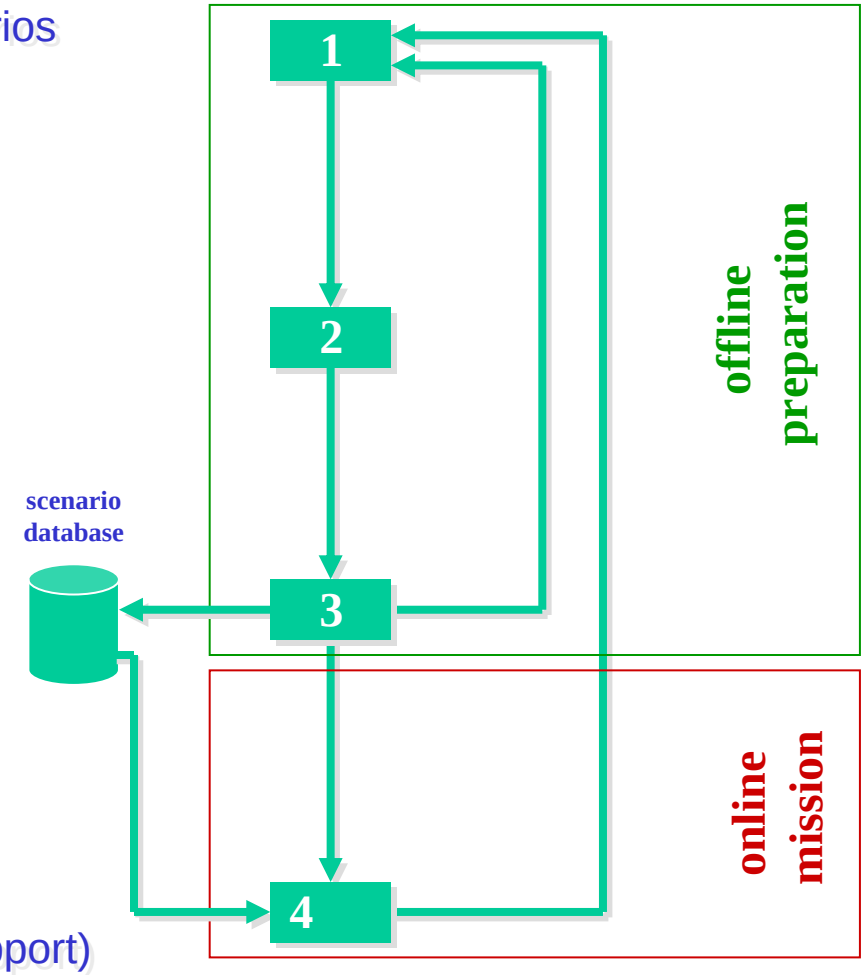


- C⁴I with szenario data base
- more coherent behavior
- „no improvisation needed“

Modes of use Combat Support

Data Farming

1. Multi-scale simulation of scenarios
   - environment
   - communication
   - reporting structures
   - order structures
   - effects
2. Analysis of variables
   - communication load
   - risk measured
   - probabilities of failure
   - …
3. Scenario analysis
   - alternative scenarios
   - critical paths
   - demands, costs, …
   - time-dependent factors
   - operation orders, bill of materials
   - training maneuvers
   - …
4. Online-C2System (decision support)



- **comparison between stored and current mission graph based on similarity measures.**

Outlook

- Scalable in space, and functionality
- Finishing the LAMPSSys-Core
- Finishing of the HLA-Impementation
- Adapter for statistical Analysis and GIS

Graph-Metric

- In a simulation the time is run in disc cycles

$$t_1 \rightarrow t_2 \rightarrow ... \rightarrow t_m$$

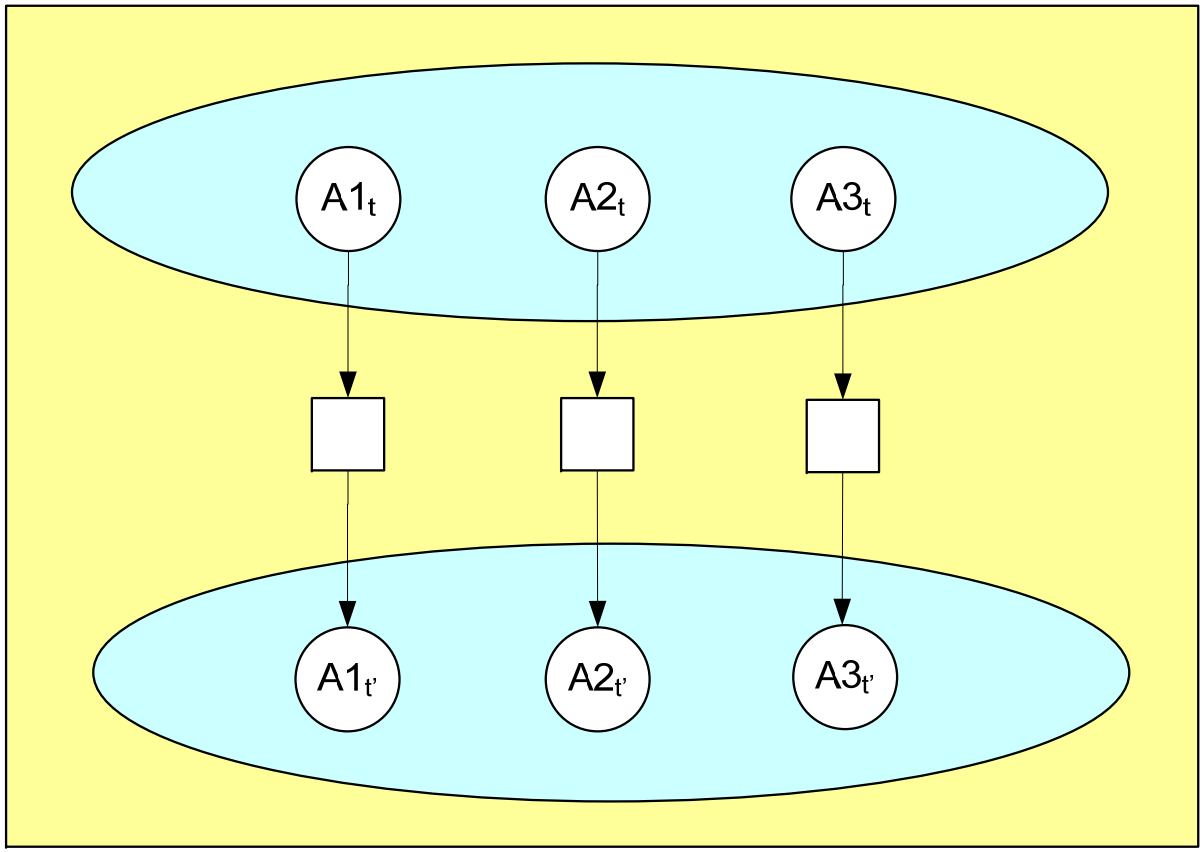
- A state S in a point in time t is described by the states of all the individual agents

$$S_t = \{s_1, s_2, ..., s_n\}$$

- A simulation run is fully described by the set of all states S in all time steps

$$R = \{r_1, r_2, ..., r_h\}$$

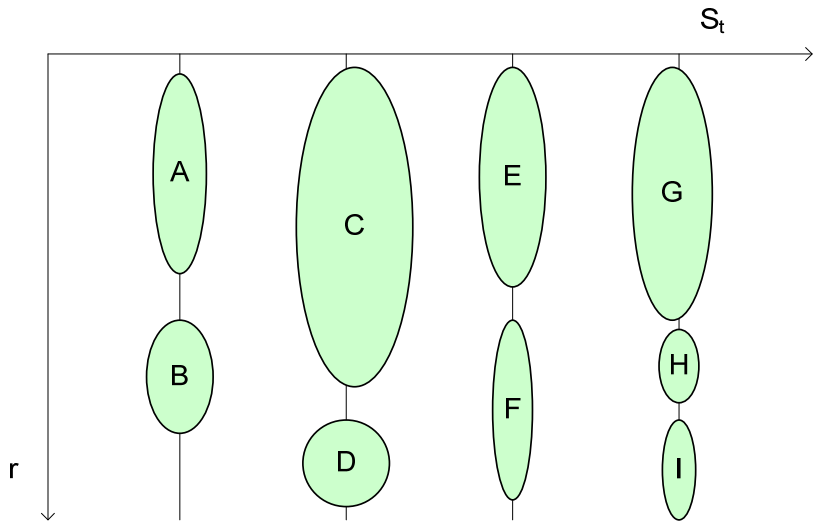
Graph-Metric



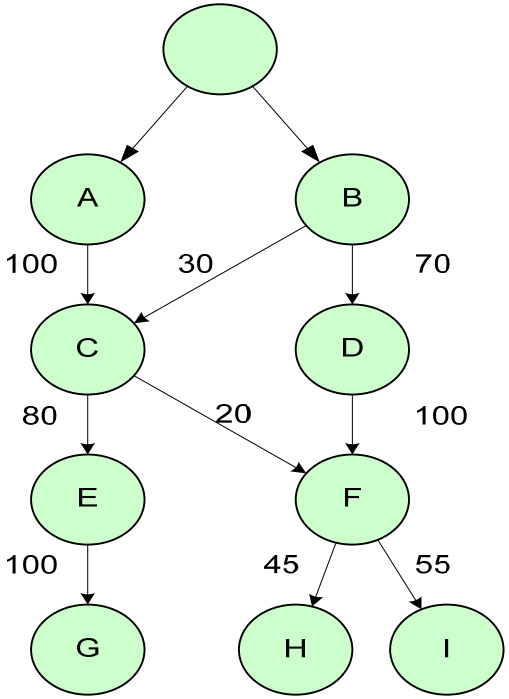
Graph-Metric

- For all attributes of all agents in all time steps this is an overwhelming amount of information
- => we pick interesting attributes beforehand:
  - The position (in the 3D space)
  - The loss of own troops
  - The time to accomplish the mission goal
  - The amount of effects to act upon the opponent

Graph-Metric



Clusters over time and runs



Markov tree with
propabilities at the edges

Technical Realization Agent Technology

- The world consists of concurrent interacting agents (also non-human parts of the game are modeled as agents)
- The real-world behavior is simulated in detail on a computer as an artificial reality
- Humans and real systems outside the simulator are coupled with this

  artificial reality by special communication agents

Thank you for your attention !